NPS-55HD71121A

NAVAL POSTGRADUATE SCHOOL

Monterey, California

NATURAL LANGUAGE INPUTS TO A
SIMULATION PROGRAMMING SYSTEM
-
AN INTRODUCTION

by

George E. Heidorn

December 1971

Approved for public release; distribution unlimited.

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral A.S. Goodfellow, USN
Superintendent

M.U. Clauser
Provost

ABSTRACT:

A simulation programming system with which models for simple queuing problems can be built through natural-language interaction with a computer is described. In this system the English statement of a problem is first translated into a language-independent entity-attribute-value information structure, which can then be translated back into an equivalent English description and into a GPSS simulation program for the problem. This processing is done on an IBM 360/67 by a FORTRAN program which is guided by a set of stratified decoding and encoding rules written in a grammar-rule language developed for this system. A detailed example of the use of the system is included.

This task was supported by the Information Systems Program of the Office of Naval Research as Project NR 049-314, under Project Order PO 1-0177.

The facilities of the W.R. Church Computer Center were utilized for this research.

TABLE OF CONTENTS

I. INTRODUCTION

In the last decade computer simulation has become a widely used tool of management science. Because of the basic similarities among computer simulation models, it was recognized early that it would be helpful to have special purpose computer programming systems designed specifically for simulation, and many such systems have been developed. The two which appear to be the most popular in the United States are GPSS [1] and SIMSCRIPT [6]. Each of these offers the user a conceptual framework within which to build his model (sometimes called the "world view"), a language in which to express the model, and a set of computer routines for executing the model.

Although the availability of these simulation programming systems has considerably reduced the task of producing simulation models, their use still requires the services of somebody trained in computer programming. It would seem that there would be some advantage to automating this part of the task. One approach suggested for accomplishing this is called "programming by questionnaire" [2], in which the computer itself writes a simulation program from answers (essentially numeric) supplied to a questionnaire. The original implementation of this scheme was a system capable of producing SIMSCRIPT programs for simulating job shops. How-

ever, it does not appear that this technique has been widely used.

In the last several years, in the field of Artificial Intelligence much effort has been devoted to the development of "natural-language question-answering systems," computer programs that will accept facts and answer questions given in English. Many papers have been written on this topic (e.g. [7, 8, 9]). Also, in the field of Linguistics a great deal of work has been done on formally specifying natural-language communication processes (e.g. [3, 4, 5, 10]). Up to now, there has been no mention in the literature of any attempt to apply any of this natural-language research to the simulation programming problem, however.

A. OBJECTIVE

The objective of the research presented in this report is to develop a simulation programming system with which an analyst can build models through natural-language interaction with a computer. The purpose of such a system is to automate the computer programming part of simulation modeling. The initial version of this system is limited to dealing only with fairly simple queuing problems.

B. ORDER OF REPORTING

In this report the overall approach used will be discussed

first, especially the linguistic considerations. Then the computer system developed will be described, followed by a detailed discussion of its use for a sample problem. Finally, there is a section of concluding remarks.

This report is intended to serve as an introduction to the system which has been developed, and, therefore, many details have been omitted. Additional reports with these details are forthcoming.

II. THE APPROACH USED

If a simulation programmer were given a queuing problem stated in a natural language, he would probably read it one or more times to form a mental image of the system being described and to note the points of interest in it. If the description were not clear to him or if essential information were missing, he might ask questions of the writer until he felt that he completely understood the problem and had all the information he needed to do the program. At this point he might state the problem "in his own words" to the writer as a check on his understanding of it. Finally, he would think about the problem in terms of the concepts of the computer language he planned to use, and then he would write the program.

The computer system developed in this research serves the same role as the simulation programmer described above. Therefore, it was designed to follow essentially the same overall procedure as he does. In this section of the report the computer's counterpart of the programmer's mental image, the Internal Problem Description, will be discussed first, followed by a discussion of some linguistic considerations of the research.

A. THE INTERNAL PROBLEM DESCRIPTION

The Internal Problem Description (IPD) is an entity-

attribute-value data structure for holding information about a particular problem in a language-independent form. Entity-attribute-value data structures have been widely used both in artificial intelligence applications and in simulation programming systems such as SIMSCRIPT and GPSS. In the IPD an entity is represented by a "record", which is just a list of attribute-value pairs. Some of the records in an IPD represent physical entities, such as a car or a dock, and others represent abstract entities, such as an action or a function. The attributes which a record has depend, of course, upon the entity being represented. The value of an attribute may simply be a number or a name, or it may be a pointer to another record.

A queuing problem typically deals with physical entities, such as cars or ships, moving through a system to be serviced in some manner at other physical entities, such as a pump or a dock, in the system. Here, the former of these are termed "mobile entities", and the latter are called "stationary entities". (In SIMSCRIPT these are temporary and permanent entities, and in GPSS they are transactions and facilities and storages.) As the mobile entities move through the system, they engage in "actions" at the stationary entities. Some of these actions are instantaneous, such as arrive and leave, and are called "events"; others, such as service and load, consume

time and are referred to as "activities" here.

The IPD describes the flow of mobile entities through a system, by specifying the actions which take place there. Each of these actions is represented by a record which has attributes to furnish such information as the type of action, the entity doing the action (i.e. the agent), the one to whom the action is being done (i.e. the goal), the location where it happens, how long it takes, how often it occurs, and what happens next. For example, the action "The men unload the ship at a dock for eight hours" could be represented by the record

Type	unload
Agent	men
Goal	ship
Location	dock
Duration	8 hours

where the values of at least some of these attributes are actually pointers to other records in the IPD, such as records to represent the men, ships, and docks. If an attribute such as duration were specified as a probability distribution, there would be a record in the IPD to represent that particular distribution, also. The Internal Problem Description for the example problem in Section IV of this report will be discussed in detail there.

B. LINGUISTIC CONSIDERATIONS

The task of translating between an Internal Problem

Description and a textual description of the same problem, in either a natural language or a programming language, falls into the realm of Linguistics. A language theory well suited for the application being described here is Stratificational Grammar [4, 5]. In this theory, language is considered to be a system existing in the brain for translating information in the form of text, which is one-dimensional, into equivalent information in the form of a multi-dimensional network in the mind of the receiver, and vice versa. The first of these two processes (i.e. text-to-network) is called "decoding", and the inverse process (i.e. network-to-text) is called "encoding". The main feature of Stratificational Grammar which distinguishes it from other language theories is that these processes are considered to consist of several levels (strata), each of which can be described separately, but in a similar fashion. The idea is that by describing the processes at each of these levels separately, the overall description of the language can be simplified.

In the work being reported on here a three-level system is considered. The "morphology" deals with the manner in which characters are put together to form parts of words and parts of words are put together to form words. The "lexology" deals with the way in which words form phrases, phrases form clauses, and clauses form sentences. And, the "semology" is

concerned with the relationship between the information in a sentence and the information in some particular portion of the Internal Problem Description.

A simple example will serve to illustrate the points discussed above. Either of the following two sentences could appear in an English description of a queuing problem:

The men unload the ship at a dock for 8 hours.
The ship is unloaded for 8 hours at a dock by the men.

At the morphological level the only really significant difference between the two is the form of the verb "unload". Adding "ed" to a verb stem to form the past participle is considered to be a morphological process. Also, there are two additional words in the second sentence ("is" and "by").

At the lexological level the two sentences are quite different, however. Each has a different subject ("men" vs. "ship") and a different ordering of the modifying phrases. A typical process in the lexology is the one which puts a form of "be" together with a past participle to form a passive verb phrase (e.g. "is unloaded"). Other processes at this level would do such things as recognize what each prepositional phrase is for (i.e. location, duration, etc.).

At the semological level the two sentences given above are identical. They have exactly the same meaning and, therefore, would be related to exactly the same structure in the

IPD. During decoding, after the completion of lexological processing, either of these sentences would be represented by a record identical to the one shown in the example in part A of this section. It would then be the task of the semological processing to merge this information properly into the IPD.

The information given by either of the above sentences could also be given by a series of shorter sentences, not necessarily contiguous in the text. For example,

> The men unload the ship.
> Unloading takes place at a dock.
> The time to unload the ship is 8 hours.

Because of the semological processing that would be done for these sentences, the resulting action record in the IPD would be identical to the one already shown.

Sentences which occur in natural language descriptions of queuing problems can be considered to fall into two categories: "action sentences" and "attribute sentences". An action sentence has as its main verb an action verb, which is modified by phrases and clauses to specify the values of the attributes of the action. For example, "After arriving, if the dock is available, the ship is unloaded at the dock" is an action sentence; the action is "unload", its goal is "ship", its location is "dock", its predecessor is "arrive", and its condition is "dock available". It should be noted

that the order of most of the phrases and clauses in this sentence could be changed without altering the information content.

An attribute sentence has as its main verb an attribute verb, and is used to specify the value of some attribute of some record in the IPD. For example, "The time to unload the ship is 8 hours" says that the value of the "time" (actually duration) attribute of the action record "unload ship" is "8 hours". An equivalent statement would be "It takes 8 hours to unload the ship."

A detailed explanation of the processing of sentences such as these is beyond the scope of this report. However, some information about the way in which decoding and encoding processes are specified to the computer is included in the next section.

III. THE SYSTEM DEVELOPED

The computer system developed to meet the objective of this research is in the form of a 4000-statement FORTRAN program called NLP (Natural Language Processor), which is intended to be useful for a wide range of natural-language, man-machine communication tasks. When run under the CP/CMS time-sharing system on an IBM 360/67, it requires a virtual machine with 350K bytes of storage. The program consists of about 100 routines, ranging in size from one which simply unpacks a four-byte word to another which is a compiler for a grammar-rule language. One large group of routines provides list-processing capabilities. The main routine serves as a monitor to provide for interaction with the user, as will be demonstrated by the example in the next section.

The bulk of the information maintained during the running of the program is in a one-dimensional array called CELL. This variable is currently DIMENSION'ed to have 19000 elements, each of which consists of eight bytes. Each element of CELL is capable of holding the "name" and value of one attribute, and all information there is in the form of records (i.e. "linked lists"), which are manipulated by the list-processing routines.

When the NLP program is first loaded, the CELL array contains no information; all of its elements are simply linked

to form a free-storage list. Before a queuing problem can be processed, the CELL array must contain information about the relevant words and concepts and about the grammars of the languages to be used (currently English and GPSS) and how text is to be processed for these languages. Information about words and concepts is entered by means of "named record" definitions, and the grammars and processing are specified by "decoding rules" and "encoding rules". Each of these three types of input will be discussed briefly in this section of the report. More detailed coverage of this material will be left for later reports.

A. NAMED RECORDS

A named record is just a record that has a NAME attribute, with a character string value of eight or fewer EBCDIC characters which is considered to be the name of the record. A named record is defined by giving its name, followed by the values of its attributes in parentheses. This information is usually punched on cards.

For each word that is to be recognized during decoding there must be a corresponding named record with information about that word and about whatever concept may be associated with it. For example, a typical definition would be

```
SERVIC ('ACTIVITY',E,ES,ING,ED,TRANS,AGORGL='GOAL')
```

which could be loosely interpreted as saying that the concept

SERVIC is in the set ACTIVITY, the verb-stem SERVIC can take the endings E, ES, ING, and ED, the verb SERVIC is TRANSitive, and the mobile entity in a SERVIC action would be the GOAL of the action.

Whenever a name appears in single quotes in any definition or rule, it is considered to be the name of a named record. Whenever such a name appears by itself, like 'ACTIVITY' in the example above, it is equivalent to SUP='...', e.g. SUP= 'ACTIVITY', where SUP stands for "superset"; in other words, the value of the SUP attribute of the named record 'SERVIC' is a pointer to the named record 'ACTIVITY'.

The SUP attribute is used to bring related concepts together in a hierarchical fashion. For example, the SUP of 'ARRIV' and 'LEAV' is 'EVENT', the SUP of 'SERVIC', 'LOAD', 'UNLOAD', etc. is 'ACTIVITY', and the SUP of both 'EVENT' and 'ACTIVITY' is 'ACTION'. A superset structure is imposed on most other concepts in a similar manner. The existence of these structures makes it possible to simplify considerably the expression of some complicated decoding and encoding processes. The basic idea of the SUP has appeared previously in the literature [8].

There is one record in the system whose name is predefined. It can be referred to as either MEMORY or MEM (without quotes).

When a named record definition is processed by NLP, an

appropriate record is created in the CELL array, with the specified attributes. Currently, there are about 300 of these records, and they require close to 2000 cells of the array. It takes about 30 seconds of computer time to process these definitions.

B. DECODING RULES

Decoding rules are used to specify to NLP the manner in which input text is to be processed to produce the Internal Problem Description. These rules are grouped by strata; there is the morphology, the lexology, and the semology.

A typical rule in the English decoding morphology is:

```
VERBS(ED)  E  D  --> VERBP(SUP(VERBS),PASTPART,PASTF)
```

This rule could be loosely interpreted as saying that if in the input stream there is a verb-stem segment with an ED attribute (to indicate that it can take an "ed"), followed by an "e", followed by a "d", then put these three segments of text together to form a verb-part segment which has the same SUP as the verb-stem and is marked as being a past-participle and a simple-past-form. (The SUP might be a pointer to the named record 'UNLOAD', for instance.)

A typical rule in the English decoding lexology is:

```
VERB('BE')  VERBPH(PASTPART)   -->
            VERBPH(PASSIVE,VFORM=VFORM(VERB))
```

This rule says that any form of the verb "be" can be put together with a past-participle verb-phrase to create a new verb-phrase that has all the characteristics of the old verb-phrase, except that it is passive and has the same verb-form (e.g. present-third-person-singular) as the verb on the left. (e.g., This rule would apply to the phrase "is unloaded".)

As can be seen from the above examples, a decoding rule consists of a list of segment types on the left of an arrow to indicate which types of contiguous segments can be put together to form a segment of the type on the right of the arrow. Conditions which must be satisfied in a segment may be stated in parentheses on the left side of the rule, and actions to be performed when a new segment is created may be stated in parentheses on the right side. A great variety of these condition specifications and creation specifications are available in this grammar-rule language. There have been two reports in the recent literature of schemes bearing some resemblance to this, but they were developed independently [3, 10].

Currently, there are about 300 English decoding rules for this application. They are punched on cards, and take about 2 minutes of computer time to "compile." Compilation converts them into equivalent information in the CELL array, where they require about 6000 cells.

If it were desired to state queuing problems to the system in a language other than English, a set of decoding rules for that language would have to be written and supplied to the computer. The rules in the new semology would be essentially the same as those in the English semology, however, because the rules in that strata have a high degree of language-independence.

C. ENCODING RULES

Encoding rules are used to specify to NLP the manner in which the Internal Problem Representation is to be processed to produce output text. Currently, there is a set of encoding rules for English and a set for GPSS, each of which is grouped by strata, similarly to the decoding rules just discussed.

A typical rule in the English encoding lexology is:

```
VERBPH(PASSIVE)  -->
       VERB('BE',VFORM=VFORM(VERBPH))
       VERBPH(-PASSIVE,-VFORM,PASTPART)
```

This rule says that a passive verb-phrase is to be expanded to a verb which is a form of "be" (with the particular verb-form coming from the verb-phrase), followed by a new verb-phrase which has all the characteristics of the old verb-phrase, except that it is not passive and it has past-participle in place of its old verb-form.

A typical rule from the English morphology is (as might be expected):

VERBP(PASTPART) --> VERBS(SUP(VERBP)) E D

This rule specifies that a past-participle verb-part is to be realized as a verb-stem with the same SUP, followed by an "e", followed by a "d". This particular rule could be considered as sort of a default to be applied in the case when none of the rules for irregular verbs is applicable.

As can be seen from the examples, an encoding rule has the name of one segment type on the left and a list of segment types on the right. The conditions specified in parentheses on the left help to determine if a rule is applicable, and the creation specifications given in parentheses on the right determine the characteristics of the segments created. The condition and creation specifications available for encoding rules are the same as for decoding rules. A scheme for producing meaningful text which bears a slight resemblance to this is mentioned in a paper by Simmons, et al [8].

The current set of encoding rules for producing English descriptions of queuing problems consists of fewer than 200 rules, and the set for producing GPSS programs is about half as big. Compilation for both of these sets together takes about 2 minutes and results in approximately 6000 cells of information being put into the CELL array.

Just as in decoding, if it were desired to have the system produce natural-language problem descriptions in another language, another set of encoding rules would have to be written. However, the current "English" semology would be used exactly as it is for many other languages; only the lexology and morphology would have to be rewritten. Similarly, simulation programs could be produced in another language, such as SIMSCRIPT, by writing an appropriate set of encoding rules. Because of the basic structural differences among these programming languages, however, the rules of all three strata would have to be rewritten. It should be noted that the languages used for input and output in this system need not be the same.

IV. AN EXAMPLE OF THE USE OF THE SYSTEM

In this section of the report a complete example will be presented to illustrate the use of the system developed in this research. The discussion consists primarily of an explanation of the information which appears in Figures 1 through 7, all of which are grouped together at the end of the section for ease of cross reference. First, stating the example problem in English is described, including the details of "getting on and off" the system. Then the Internal Problem Description for this particular problem is discussed, as is its development. Finally, the encoding of an English problem description and the encoding of a GPSS program for the problem are described in some detail. Computer timing information is given for each portion of the process, also.

A. STATING THE PROBLEM IN ENGLISH

Figure 1 shows part of the first of two terminal sessions used to produce the example being presented here. All upper case typing was done by the computer, and all lower case typing was done by the user. The first line is the command to load the program. The routines of this program are grouped into six files, the names of which are listed after "load".

The next line is a standard line produced by the time-sharing system to indicate that it is ready for another command. Included is information about the time taken to do the task just completed -- both virtual CPU time and actual CPU time (including overhead for paging, etc.) -- and the time of day. In this case it took 1.70 seconds of virtual CPU time and 4.02 seconds of actual CPU time to load the program, and the time of day was 41 seconds past 3:10 PM.

The "start" command begins execution of the program which has just been loaded, as can be seen by the message produced by the time-sharing system. The next message comes from NLP, and gives the user an opportunity to change the values of some preset parameters in the program which control the amount of output and where it appears. For this session the default values of these parameters were desired, so all that had to be entered were the input delimiters "&p" and "&end".

Then NLP requests the number of a file in cell structure format produced by a previous run of the program. (A response of 0 would mean to "start from scratch".) In this case file 9 was specified because it contained the cell structure previously produced by processing the named records and the decoding and encoding rules. After reading file 9 into the CELL array, the program has all of the information

it needs to "understand" an English language description of a queuing problem presented to it, and to later produce its own description of the problem and a GPSS program for it.

Then the program requests the number of a file in card-image format to tell it what to do next. This could be a file read offline from the card reader, or it could be the terminal. In this case a response of "t" was given to specify that input is to be obtained from the terminal. The first line of input must contain a command to tell the program what to do. All commands to NLP must end with a colon. Currently, there are almost twenty commands available to invoke the various routines in the system. Some of these expect additional lines of input, and some do not.

The "decode" command given here causes the DECODE subroutine to be called. This routine reads text from the input stream and applies the decoding rules to it. The remaining nine lines in Figure 1 were processed in this way. Each sentence was started on a new line just for clarity in the figure; this is not required by the program. The triple spacing comes about because the program spaces once to indicate that a line has been read and then once again to indicate that it is ready to read another line. The circled sentence numbers were added to the figure for later reference. As each sentence is processed, the information extracted from it is en-

tered into the Internal Problem Description being constructed. This will be discussed in some detail in parts B and C of this section.

Figure 2 shows the rest of the first terminal session. The first line is a message from the operator reminding the user that it was almost 4:00 PM, the time at which the time-sharing service terminates for the day. So, it was necessary to save what had been done and "get off". First, a double colon was typed, to signify an end-of-file to NLP. Then, when the number of the next input file was requested, a response of 0 was given. This particular response brings about the message requesting the number of an output file. Responding to this with a 7 caused the current cell structure to be written out into file 7. At this time a message is also printed to inform the user about the maximum number of cells which have been used at any point in the processing that has been done and about the number of cells currently being used to hold all of the information (including the rules, etc.). In this case, the numbers were 17211 and 14217, respectively. From this it can be seen that approximately 3000 cells were required temporarily at some point during the decoding process, probably for the third sentence. Both MAXLN and UCELLS had values of about 14000 for file 9 at the beginning of this session.

Again the program requests the number of an input file. A response of "t" was given, followed by the command "end", to terminate execution of NLP. The Ready line produced by the time-sharing system shows that 96.42 seconds of virtual CPU time and 212.34 seconds of actual CPU time were used by this execution of the program. From the time of day, it can be seen that about 50 minutes had elapsed since the program was loaded. This high ratio of elapsed time to CPU time is caused partly by the paging characteristics of this program and partly by a heavy load on the time-sharing system.

Figure 3 shows the beginning of the second terminal session used to produce the example being presented here. It is actually quite similar to Figures 1 and 2 combined. It can be seen that this time the initial input file was 7, the one that had been written at the end of the previous day. After the decode command, three more sentences were entered to complete the specification of the example problem. Then the cell structure with this additional information was written back out into file 7, replacing the old file 7. By comparing MAXLN and UCELLS with the corresponding values in Figure 2, it can be seen that the maximum number of cells used during the processing had not changed, but the number of cells currently being used had increased by 41, due to the additional information. These 41 cells would all be part of the IPD.

Again execution of NLP was terminated by the end command. The Ready line shows that virtual CPU time for this run was 45.42 seconds and actual CPU time was 102.74 seconds. Also, it can be seen that elapsed time was about 19 minutes, resulting in a lower ratio of elapsed time to CPU time than on the previous day, probably due to a lighter load on the time-sharing system. Combining this timing information with that obtained on the previous day shows that for NLP to decode the English statement of this example queuing problem into an Internal Problem Description required about 2½ minutes of virtual CPU time, about 5 minutes of actual CPU time, and a little over an hour of elapsed time at the terminal.

B. THE INTERNAL PROBLEM DESCRIPTION

A graphic portrayal of the IPD for this example problem appears in Figure 4. Information needed to do this drawing was obtained by making another run of the program, using 7 as the initial input file and then giving a series of print commands (e.g. "print 'actnlist',2:"). The actual run is not included here.

In the figure each record of the IPD is represented by a box, with the name of the record appearing at the top of the box. With the exception of MEMORY and 'ACTNLIST', these names do not actually exist within the computer, but were placed on the drawing simply to furnish a means of referring

to the various records in the discussion which follows. In each box the attribute-value pairs of the record are shown, with the attribute name or number on the left and its value on the right. Many of the values are pointers to other records in the IPD, in which case an appropriate arrow is drawn.

It can be seen that MEMORY is the only record which is not pointed to by some other record. It plays a rather central role in the IPD, being used both to hold global information about the problem (e.g. problem time and the basic time unit) and to serve as sort of a directory into the rest of the IPD. Only one portion of the "directory" was included in this drawing in order to keep the number of lines at a minimum. The portion included is the "action list" ('ACTNLIST'), which, as can be seen, contains pointers to each of the three action records. Not included in the drawing are the lists for mobile entities ('MOBLIST'), stationary entities ('STALIST'), distributions ('DSTRLIST'), and successor descriptors ('SCSRLIST'). The action list may be considered to be the most important list, because of the key role which actions play in a problem description.

Every IPD record, except for MEMORY and the lists just mentioned, has a SUPerset attribute pointing to the named record representing the concept of which this record is a specific instance. For example, the SUP attribute of the first

action record (REC11) points to the named record 'ARRIV', indicating that this action (vehicles arrive at a station) is a specific instance of the concept "arriv".

Each action record in the IPD must have either an AGENT or a GOAL which points to a mobile entity record. The AGENT of an action is the one doing the action, and the GOAL is the one to whom the action is being done. The MTR attribute tells which of these two is pointing to a mobile entity. Each action record must also have a LOCATION attribute pointing to a "location descriptor" record, which in turn points to a stationary entity record. An event like 'ARRIV' or 'ENTER' must have an IETM (inter-event time) attribute to specify the time between occurrences of the event, and an activity (e.g. 'SERVIC' or 'LOAD') must have a DURATION attribute to specify the time taken to perform the activity. These times can be given as constants, standard probability distributions, functions, or combinations of these, some of which can be seen in the drawing. REC42 in the drawing is a function which has the records for car and truck as its X values and the records for 5 minutes and 9 minutes as its Y values. The ASNDISTR attribute of an 'ARRIV' specifies the percentages of the various kinds of entities which arrive, in the form of a cumulative probability distribution. REC43 in the drawing furnishes an example of this. (The NUM attri-

bute of a 'DECIMAL' record is considered to be in parts-per-thousand.) The attributes DORC, FNARG, and PNUM which appear in REC42 and REC43 are needed for encoding the GPSS program.

Each action record, except a 'LEAV', must have a SUCCessor attribute to specify which action the mobile entity of this action is involved in next. The value of SUCC may simply be a pointer to another action record, or it may be a pointer to a "successor descriptor" record. REC51 in the drawing is an example of one of the five types of successor descriptors currently available in the system. This particular record, which is a 'QTYP', can be interpreted as saying, "If the length of the line at the pump (SUCARG) is less than two (MAXQ), go to be serviced (OPENACT); otherwise, leave (CLOSACT)." The other types of successor descriptors available handle such situations as "If the pump is busy, the vehicle leaves.", "Cars are serviced, and trucks leave.", and "Half of the vehicles are serviced, and the rest leave."

It can be seen in the drawing that the records for 'CAR' and 'TRUCK' each have a STRUCture attribute pointing to the record for 'VEHICLE'. This is related to the idea of the "assignment distribution" (ASNDISTR), and essentially means that cars and trucks may be referred to as vehicles in the problem description. The attribute CLASATR (class attribute) in the 'VEHICLE' record indicates what is the distinguishing

attribute of any records which have a STRUC attribute pointing to this record. (SOUP is synonymous with SUP in this case.) Part of the usefulness of the STRUC attribute is that it avoids some unnecessary duplication of information. For example, the value of the CONSUMPtion attribute is the same for both cars and trucks in this problem, so it need be stored only once, up in the 'VEHICLE' record. (CONSUMP indicates how many units of a resource are required by a mobile entity.) Each entity and action record is assigned an identification number (IDNO) for use in the GPSS program.

C. DEVELOPING THE INTERNAL PROBLEM DESCRIPTION

Figure 5 is included to help the reader relate the English description of the problem which the user entered, shown in Figures 1 and 3, to the Internal Problem Description shown in Figure 4. Basically what this figure shows is when each record in the IPD was created and when each attribute was given its value. The leftmost column in the figure contains sentence numbers, the numbers which appear in circles in the earlier figures. The next two columns give the names of the records created or changed (i.e. given additional attribute values) when each sentence was processed by the decoder. The attributes given values at a particular time are listed to the right of the record name. For example, the figure shows that when sentence 1 ("Vehicles arrive at a

station.") was processed, REC21 and REC31 were created and their SUP and IDNO attributes were given values. Also, REC61 was created, with values for its SUP and LOCOBJ attributes, and REC11 was created, with values for its SUP, IDNO, AGENT, and LOCATION. Then, when sentence 3 ("... after arriving....") was processed, the SUCC attribute was added to REC11.

Although it is not shown in Figure 5, the list records were affected by the creation of some of these records, also. For example, when sentence 1 was processed, the LASTREC attributes of 'MOBLIST', 'STALIST', and 'ACTNLIST' were incremented from 10 (their initial value) to 11, and attribute 11 of each was set to point to the newly created records (REC21, REC31, and REC11, respectively). This information was left out of the figure so as not to clutter it unnecessarily. It should also be noted that the order in which record names appear for a particular sentence may not be exactly the order in which the records were created. Usually a record is created with just a SUP at some point in the processing of a sentence, and then other attributes are added to it as more of the sentence is decoded.

A careful comparison of Figures 4 and 5 will reveal that there are some attributes shown in the IPD which were not given values during decoding. These are ones which are needed for producing a GPSS program, and actually would get

their values as part of the GPSS encoding process. CAPACITY, QUANTITY, and CONSUMP are given default values of 1 or 'ONE' if they are not specified in the original problem description. ('ONE' is a named record, with a SUP of 'UNIT' and a NUM of 1.) The value of IDNAME is formed at that time by concatenating the first three or four letters of the NAME of the SUP of a record with the value of its IDNO. Also, during both decoding and encoding, a number of attributes needed temporarily for the processing "come and go" in the IPD records.

D. ENCODING THE ENGLISH PROBLEM DESCRIPTION

After the NLP program was run to print out the information about the Internal Problem Description, another run was made to encode the IPD into an English description of the problem and into a GPSS simulation program for the problem. The first part of this run is shown in Figure 6. It can be seen that this run began the same as others already described. However, in this one, after the decode command was given, instead of stating some information about the problem, a command in the form of an English sentence was given. The decoding of that sentence resulted in a call to the encoder being made to produce the English problem description. (Actually, the same result could have been obtained at the NLP command level by entering "encode english:", but that would not be quite so conversational.)

The overall manner in which the English description is produced can be seen by comparing the information in the text with the information in the IPD. The first paragraph is produced by going down the action list and saying something about the attributes of each action. The very first action is simply stated with a simple sentence containing information about the type of action, its AGENT and/or GOAL, and its LOCATION. If the IETM or DURATION attribute has a simple value, it will be included also, as a prepositional phrase (e.g. "every 8 minutes" or "for 5 minutes"). Otherwise, a separate statement will be made about the IETM or DURATION, as can be seen in the figure. If the action has an ASNDISTR, a statement will then be made about it, as also can be seen in the figure. Finally, a statement of the form "After ..., ..." is produced from the SUCC attribute. The exact form of this statement depends upon the type of value which SUCC has. It can be seen in the figure that a 'QTYP' successor descriptor actually results in two sentences, with the first one having an "if" clause and the second one beginning with "otherwise".

When describing an action which has already been mentioned in a successor statement, it is not necessary to produce a simple sentence about that action. If the action has a non-

simple DURATION and/or a SUCC, the appropriate statements about these can immediately be made. This is the case for the 'SERVIC' action in the example. When the 'LEAV' action was reached in the scan of the action list, actually no output was produced from it, because it had already been mentioned a couple of times in successor statements and it had no additional attributes to be described. If a stationary entity has a QUANTITY or CAPACITY attribute with a value greater than 1, a statement will be made about it shortly after the entity is first mentioned in an action sentence (e.g. "There are 2 pumps in the station." or "The capacity of the station is 8 vehicles."). After describing the actions and the entities, a separate one-sentence paragraph is produced with the values of PROBTIME and TIMUNIT of MEMORY, as can be seen in the figure.

Although timing information does not appear in the figure, the virtual CPU time required for NLP to encode the IPD into English text was 25 seconds, the actual CPU time was 76 seconds, and the elapsed time was about 13 minutes.

E. ENCODING THE GPSS PROGRAM

After the English problem description was produced, NLP was ready to accept another sentence to be decoded. At this time another command in the form of an English sentence was entered, as can be seen in Figure 7. This resulted in a call

to the encoder, which produced the GPSS program shown there. ("encode gpssprog:" would have accomplished the same thing at the NLP command level.)

The manner of producing the GPSS program is similar to that for the English description, but it involves going down several lists, not just the action list. As was mentioned earlier, these other lists are not actually shown in the IPD drawing in Figure 4. Their contents will be given in parentheses at appropriate points in the following discussion, however. The first bit of output the GPSS encoding rules produce is a standard SIMULATE card and RMULT card. Then a pass is made down the stationary entity list (REC31, REC32) to produce an EQU card for each stationary entity, to relate its IDNAME and its IDNO and to define it as a facility or a storage and a queue. If either the QUANTITY or CAPACITY attribute is greater than 1, an appropriate STORAGE definition card is also produced. Then a similar pass is made down the mobile entity list (REC21, REC22, REC23) to output an EQU card and a TABLE card for each type of mobile entity that will actually appear in the simulation (i.e. those records that do not have a CLASATR attribute). In the example, nothing is included for 'VEHICLE' because any vehicle that appears is either a car or a truck. The tables defined will be used to record transit times during the simulation.

Next, a standard FUNCTION 1 for the exponential distribution and a standard FUNCTION 2 for the unit normal distribution are produced if they are required by the problem. Then a pass is made down the distribution list (REC41, REC42, REC43, REC44) to define a FUNCTION for each record that requires one. In the example, FUNCTION 3 comes from REC42, and FUNCTION 4 comes from REC43. This is followed by a similar pass down the successor descriptor list (REC51) to define a FUNCTION for each record that requires one. This pass produced nothing in the example. Then the records in the distribution list are looked at once again to define an FVARIABLE for each normal distribution used in the problem. One of these appears in the example. The numbers 16 and 4 appear there for the mean and standard deviation rather than 8 and 2, as might be expected, because the basic time unit to be used for this problem was specified as 30 seconds rather than 1 minute. The number of each FUNCTION and FVARIABLE defined in the above passes is stored as the IDNO attribute of the record which caused the definition, for use in later processing.

After the definitions have been taken care of, a pass is made down the action list to produce the executable blocks which describe the flow of transactions through the program (which corresponds to the flow of mobile entities through the actual system). For each action a blank comment card (with

an asterisk in column 1), followed by a comment card with a simple action sentence on it is immediately put out, utilizing a portion of the English encoding rules. This is then followed by the blocks appropriate to this action.

The group of blocks produced from an action actually has two parts, the first of which depends upon the type of action and the second of which depends upon the type of value the SUCC attribute has. For example, an 'ARRIV' usually produces a GENERATE and an ASSIGN, a 'LEAV' produces a TABULATE and a TERMINATE, and most activities produce a sequence like QUEUE, SEIZE, DEPART, ADVANCE, and RELEASE, or minor variations thereof. A 'QTYP' successor descriptor results in a TEST, followed by a TRANSFER (if necessary), and a simple SUCC results in an unconditional TRANSFER, as can be seen in the example. If the 'LEAV' and 'SERVIC' actions had been in reverse order in the action list, the resulting GPSS program would not have needed the two unconditional TRANSFER's which appear in this program, and they would have been suppressed by the encoding rules.

The contents of most of the argument fields of the various blocks depend, of course, upon the attributes of the records in the IPD. For example, argument A of the GENERATE block is V1 here because FVARIABLE 1 corresponds to the normal distribution which is the value of the IETM attribute of the

'ARRIV' action. Similarly, argument B of the ASSIGN block (which assigns the transaction type, either 2 or 3, to parameter 1 of the transaction) comes from the ASNDISTR attribute of the same action. Arguments A, B, and C of the TEST block and argument B of the TRANSFER come directly from the attributes SUCARG, MAXQ, CLOSACT, and OPENACT of the 'QTYP' record. The LOCATION attribute determines the A argument for such blocks as QUEUE, DEPART, SEIZE, and RELEASE, as can be seen in the example.

It can also be seen that argument A of the ADVANCE block (the mean advance time) references FUNCTION 3, which was defined from the 'TYPTABL' record which specifies the mean of the DURATION of the 'SERVIC' action. When a transaction enters that ADVANCE block, the appropriate mean time will be obtained from FUNCTION 3 using the value of parameter 1 which was ASSIGN'ed to it when it "arrived". This will then be modified by a value from FUNCTION 1 to yield a service time from the desired exponential distribution. The B argument of the last TRANSFER gets its value directly from the SUCC attribute of the 'SERVIC' action. All actions are referenced by names of the form "ACTi", where i is the value of the action's IDNO attribute.

Finally, after the blocks for the actions are put out, a standard "timing loop" is produced to govern the run length

of the simulation. The value in the A argument of the GENERATE block comes from PROBTIME of MEMORY. In the example this value is 960, because there are 960 30-second periods in 8 hours.

Although timing information does not appear in this figure, either, the virtual CPU time required for NLP to encode the IPD into a GPSS program was 24 seconds, the actual CPU time was 64 seconds, and the elapsed time was about 10 minutes. These times are approximately the same as those for encoding the English description.

```
load nlp prnams decode encode lpr bitstuff
R; T=1.70/4.02 15.10.41

start
EXECUTION BEGINS...
ENTER OPTIONAL DATA
 &p &end
TYPE NUMBER OF INITIAL INPUT FILE

9

TYPE NUMBER OF NEXT INPUT FILE

t


decode:

Vehicles arrive at a station.      (1)

The station has just one pump.     (2)

A vehicle will leave the station immediately after arriving   (3)
     if the length of the line at the pump is not less than two.

Otherwise, it is serviced there; then it leaves.   (4)

Service times are exponential, with a mean of 5 minutes for
     cars and 9 minutes for trucks.   (5)

Three quarters of the vehicles are cars and one fourth of them
     are trucks.   (6)
```

Figure 1. Beginning the English Statement of the Problem

```
FROM OPERATOR:  GOODNIGHT SEE YOU TOMORROW!!!!!!!!!!!!!!!!!!

::
TYPE NUMBER OF NEXT INPUT FILE
0
TYPE NUMBER OF OUTPUT FILE
7
MAXLN = 17211     UCELLS = 14217

TYPE NUMBER OF NEXT INPUT FILE
t

end:
R; T=96.42/212.34 15.59.50
```

Figure 2. Saving the Cell Structure for Part of the Problem

```
load nlp prnams decode encode lpr bitstuff
R; T=1.82/4.50 13.28.32

start
EXECUTION BEGINS...
ENTER OPTIONAL DATA
 &p &end
TYPE NUMBER OF INITIAL INPUT FILE

7

TYPE NUMBER OF NEXT INPUT FILE

t

decode:

Arrivals are normally distributed with a mean of eight minutes

     and a standard deviation of two minutes.        (7)

The simulation run time desired is eight hours.      (8)

The basic time unit to be used in the model is 30 seconds.   (9)

::
TYPE NUMBER OF NEXT INPUT FILE

0

TYPE NUMBER OF OUTPUT FILE

7

MAXLN = 17211     UCELLS = 14258

TYPE NUMBER OF NEXT INPUT FILE

t

end:

R; T=45.42/102.74 13.47.29
```

Figure 3. Finishing the Statement of the Problem

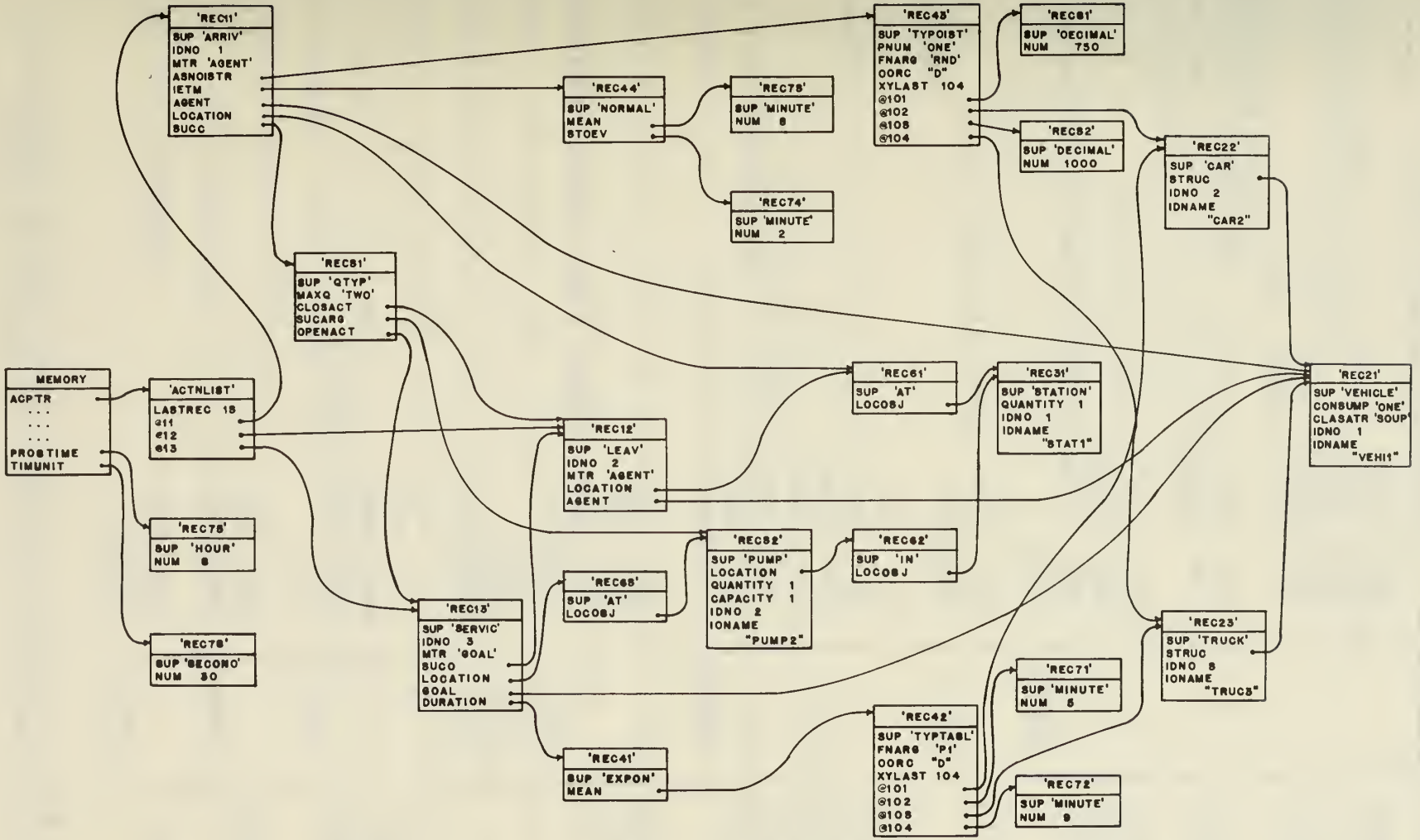

Figure 4. The Internal Problem Description (IPD)

Sentence	Record Created	Record Changed	Attributes Given Values
1	REC21		SUP, IDNO
	REC31		SUP, IDNO
	REC61		SUP, LOCOBJ
	REC11		SUP, IDNO, AGENT, LOCATION
2	REC32		SUP, IDNO, QUANTITY, LOCATION
	REC62		SUP, LOCOBJ
3	REC12		SUP, IDNO, AGENT, LOCATION
	REC63		SUP, LOCOBJ
	REC51		SUP, SUCARG, MAXQ, CLOSACT
		REC11	SUCC
4	REC13		SUP, IDNO, GOAL, LOCATION, SUCC
		REC51	OPENACT
5	REC71		SUP, NUM
	REC72		SUP, NUM
	REC22		SUP, IDNO
	REC23		SUP, IDNO
	REC42		SUP, FNARG, DORC, XYLAST, @101-104
	REC41		SUP, MEAN
		REC13	DURATION
6	REC81		SUP, NUM
	REC82		SUP, NUM
	REC43		SUP, PNUM, FNARG, DORC, XYLAST, @101-104
		REC21	CLASATR
		REC22	STRUC
		REC23	STRUC
		REC11	ASNDISTR
7	REC73		SUP, NUM
	REC74		SUP, NUM
	REC44		SUP, MEAN, STDEV
		REC11	IETM
8	REC75		SUP, NUM
		MEMORY	PROBTIME
9	REC76		SUP, NUM
		MEMORY	TIMUNIT

Figure 5. The Development of the Internal Problem Description

```
load nlp prnams decode encode lpr bitstuff
R; T=1.70/3.96 14.19.49

start
EXECUTION BEGINS...
ENTER OPTIONAL DATA
 &p &end
TYPE NUMBER OF INITIAL INPUT FILE

7

TYPE NUMBER OF NEXT INPUT FILE

t

decode:

Describe the problem in English.

      THE VEHICLES ARRIVE AT THE STATION.  THE TIME BETWEEN ARRIVALS
OF THE VEHICLES AT THE STATION IS NORMALLY DISTRIBUTED, WITH A MEAN OF
8 MINUTES AND A STANDARD DEVIATION OF 2 MINUTES.  75 PERCENT OF THE
VEHICLES ARE CARS, AND THE REST ARE TRUCKS.  AFTER ARRIVING AT THE
STATION, IF THE LENGTH OF THE LINE AT THE PUMP IN THE STATION IS LESS
THAN 2, THE VEHICLE WILL BE SERVICED AT THE PUMP IN THE STATION.
OTHERWISE, THE VEHICLE WILL LEAVE THE STATION.  THE TIME FOR THE
VEHICLES TO BE SERVICED AT THE PUMP IN THE STATION IS EXPONENTIALLY
DISTRIBUTED, WITH A MEAN OF 5 MINUTES FOR THE CARS, AND 9 MINUTES FOR
THE TRUCKS.  AFTER BEING SERVICED AT THE PUMP IN THE STATION, THE
VEHICLES LEAVE THE STATION.

      THE SIMULATION IS TO BE RUN FOR 8 HOURS, USING A BASIC TIME
UNIT OF 30 SECONDS.
```

Figure 6. Producing the English Problem Description

Write a GPSS program for this problem.

```
        SIMULATE
        RMULT       277,423,715,121,655,531,999,813
 STAT1  EQU         1,F,Q
 PUMP2  EQU         2,F,Q
 CAR2   EQU         2,T
 2      TABLE       M1,1,1,2
 TRUC3  EQU         3,T
 3      TABLE       M1,1,1,2
 1      FUNCTION    RN1,C24
0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.39/
.8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2/
.97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9997,8/
 2      FUNCTION    RN2,C29
0,-3/.012,-2.25/.027,-1.93/.043,-1.72/.062,-1.54/.084,-1.38/
.104,-1.26/.131,-1.12/.159,-1/.187,-.89/.23,-.74/.267,-.62/.334,-.43/
.432,-.17/.5,0/.568,.17/.666,.43/.732,.62/.77,.74/.813,.89/.841,1/
.869,1.12/.896,1.26/.916,1.38/.938,1.54/.957,1.72/.973,1.93/
.988,2.25/1,3/
 3      FUNCTION    P1,D2
CAR2,10/TRUC3,18/
 4      FUNCTION    RN3,D2
.750,CAR2/1.000,TRUC3/
 1      FVARIABLE   16+4*FN2
*
*       THE VEHICLES ARRIVE AT THE STATION.
        GENERATE    V1
        ASSIGN      1,FN4
        TEST L      Q$PUMP2,2,ACT2
        TRANSFER    ,ACT3
*
*       THE VEHICLES LEAVE THE STATION.
 ACT2   TABULATE    P1
        TERMINATE
*
*       THE VEHICLES ARE SERVICED AT THE PUMP IN THE STATION.
 ACT3   QUEUE       PUMP2
        SEIZE       PUMP2
        DEPART      PUMP2
        ADVANCE     FN3,FN1
        RELEASE     PUMP2
        TRANSFER    ,ACT2
*
*       TIMING LOOP
        GENERATE    960
        TERMINATE   1
        START       1
        END
```

Figure 7. Producing the GPSS Program

V. CONCLUSION

The initial version of a simulation programming system with which an analyst can build models through natural-language interaction with a computer has been developed, as evidenced by the example queuing problem presented in the previous section of this report. Actually, this system is just a particular application of a much more general system developed in this research which is intended to be useful for a wide variety of natural-language, man-machine communication tasks. For any particular application a set of decoding and encoding rules, along with some named record definitions, must be written to specify the processing to be done. In this case rules were written for subsets of English and GPSS sufficient to produce simulation programs for simple queuing problems stated to the computer in English.

Although the example given in Section IV is adequate for demonstrating the overall capability of the system, it does not show everything that it can do. The example was intentionally kept simple, primarily so that the entire Internal Problem Description could be shown readily. It should be noted that there is no theoretical limitation on the number of entities and actions which can appear in a problem description. It should also be noted that the English statement of the example problem entered by the user (shown in Fig-

ures 1 and 3) is only one of the many ways in which that particular problem could be stated to the system. For instance, the first paragraph of the English problem description encoded by the system (shown in Figure 6) would be acceptable as input to the decoder.

Although the system is basically quite capable, in its current form it would probably not be a very practical tool for an analyst with a queuing problem because it is limited both in the kinds of problems that it can handle and in the language which it will accept. These are not theoretical limitations, however, and with additional work a practical and useful system could be produced. As this additional work is done, further reports will be issued.

LIST OF REFERENCES

1. GENERAL PURPOSE SYSTEMS SIMULATOR/360 - INTRODUCTORY USER'S MANUAL, Publication H20-0304, IBM DP Div, White Plains, N. Y., 1967.

2. Ginsberg, A. S., Markowitz, H. M., and Oldfather, P. M., "Programming by questionnaire," MEMORANDUM RM-4460-PR, The RAND Corporation, Santa Monica, Calif., April 1965.

3. Kaplan, Ronald M., "The MIND system: a grammar-rule language," MEMORANDUM RM-6265/1-PR, The RAND Corporation, Santa Monica, Calif., April 1970.

4. Lamb, Sydney M., OUTLINE OF STRATIFICATIONAL GRAMMAR, Revised edition, Georgetown University Press, Washington, D.C., 1966.

5. Lamb, Sydney M., "Linguistic and cognitive networks," In COGNITION: A MULTIPLE VIEW (Garvin, ed.), 1970.

6. Markowitz, H. M., Hausner, B., and Karr, H. W., SIMSCRIPT A SIMULATION PROGRAMMING LANGUAGE, Prentice-Hall, Englewood Cliffs, N. J., 1963.

7. Minsky, Marvin (ed.), SEMANTIC INFORMATION PROCESSING, The MIT Press, Cambridge, Mass., 1968.

8. Simmons, R. F., Burger, J. F., and Schwarcz, R. M., "A computational model of verbal understanding," In PROC. AFIPS 1968 FJCC, Thompson Book Co., Washington, D.C., 1968, pp. 441-456.

9. Simmons, Robert F., "Natural language question-answering systems," COMM. ACM 13, 1 (January 1970), 15-30.

10. Woods, W. A., "Transition network grammars for natural language analysis," COMM. ACM 13, 10 (October 1970), 591-606.

INITIAL DISTRIBUTION LIST

	No. Copies
Defense Documentation Center (DDC) Cameron Station Alexandria, Virginia 22314	20
Library Naval Postgraduate School Monterey, California 93940	2
Dean of Research Administration Naval Postgraduate School Monterey, California 93940	2
Library Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	3
W. R. Church Computer Center Naval Postgraduate School Monterey, California 93940	2
Office of Naval Research Code 437 Information Systems Program Department of the Navy Arlington, Virginia 22217	2
Office of Naval Research Branch Office/Boston 495 Summer Street Boston, Massachusetts 02210	1
Office of Naval Research Branch Office/Chicago 536 South Clark Street Chicago, Illinois 60605	1
Office of Naval Research Branch Office/Pasadena 1030 East Green Street Pasadena, California 91101	1
Director, Naval Research Laboratory ATTN: Library, Code 2029 (ONRL) Washington, D. C. 20390	6

	No. Copies
U. S. Naval Research Laboratory Code 2000 Technical Information Officer Washington, D. C. 20390	6
Commandant of the Marine Corps (Code AX) Dr. A. L. Slafkosky Scientific Advisor Washington, D. C. 20380	1
Professor Alvin F. Andrus Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	1
Mr. Daniel G. Bobrow Bolt, Beranek and Newman, Inc. Cambridge, Massachusetts 02139	1
Professor Charles P. Bonini Graduate School of Business Stanford University Stanford, California 94305	1
Consolidated Analysis Centers, Inc. 1815 North Fort Myer Drive Arlington, Virginia 22209	1
Professor Lewis G. Creary Department of Philosophy Case Western Reserve University Cleveland, Ohio 44106	1
Professor James Emery Wharton School of Business University of Pennsylvania Philadelphia, Pennsylvania 19104	1
Mr. Alfred M. Feiler Project TRANSIM Department of Engineering Engineering I, Room 3076 University of California Los Angeles, California 90024	1

	No. Copies
Professor Robert B. Fetter Department of Administrative Sciences Yale University New Haven, Connecticut 06520	2
Professor Charles J. Fillmore Computer and Information Science Research Center The Ohio State University Columbus, Ohio 43210	1
Professor George S. Fishman Department of Administrative Sciences Yale University New Haven, Connecticut 06520	1
Professor Joyce Friedman Department of Computer and Communication Sciences University of Michigan Ann Arbor, Michigan 48104	1
Mr. Murray A. Geisler The RAND Corporation 1700 Main Street Santa Monica, California 90406	1
Professor David G. Hays Department of Linguistics State University of New York Buffalo, New York 14214	1
Mr. Martin Kay The RAND Corporation 1700 Main Street Santa Monica, California 90406	1
Mr. Philip J. Kiviat Simulation Associates, Inc. 1263 Westwood Boulevard Los Angeles, California 90024	1
Professor Sydney M. Lamb Linguistics Automation Project Yale University New Haven, Connecticut 06520	2
Professor James L. McKenney Graduate School of Business Administration Harvard University Soldiers Field Boston, Massachusetts 02163	1

	No. Copies
Professor Richard L. Nolan Graduate School of Business Administration Harvard University Soldiers Field Boston, Massachusetts 02163	1
Mr. Arnold W. Pratt Division of Computer Research and Technology National Institutes of Health Bethesda, Maryland 20014	1
Mr. Bertram Raphael Stanford Research Institute Menlo Park, California 94025	1
Mr. Julian Reitman Preliminary Design Engineering Norden Division United Aircraft Corporation Norwalk, Connecticut 06856	1
Miss Jean E. Sammet Federal Systems Division IBM Corporation Cambridge, Massachusetts 02139	1
Professor Roger C. Schank Computer Science Department Stanford University Stanford, California 94305	1
Mr. Robert M. Schwarcz System Development Corporation 2500 Colorado Avenue Santa Monica, California 90406	1
Mr. J. C. Shaw The RAND Corporation 1700 Main Street Santa Monica, California 90406	1
Professor Martin Shubik Department of Administrative Sciences Yale University New Haven, Connecticut 06520	2
Professor Robert F. Simmons Department of Computer Sciences University of Texas Austin, Texas 78712	1

	No. Copies
Professor Terry Winograd Artificial Intelligence Laboratory Massachusetts Institute of Technology Cambridge, Massachusetts 02139	1
Professor William A. Woods Computation Laboratory Harvard University Cambridge, Massachusetts 02139	1
LCDR Richard C. Hansen, USN Naval Advisory Group Box 8 (N13) F.P.O. San Francisco 96626	1
LCDR Eldon S. Baker, USN Fleet Computer Programming Center Pacific San Diego, California 92147	1
LT Robert T. McGee, USN USS Harder (SS-568) F.P.O. San Francisco 96601	1
CAPT Frederick H. Hemphill, Jr., USMC Subunit 1, MCTSSA MCAS (H) Santa Ana, California 92710	1
Professor George E. Heidorn Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	30

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R & D
(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

ORIGINATING ACTIVITY (Corporate author) Naval Postgraduate School Monterey, California 93940	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP

REPORT TITLE
Natural Language Inputs to a Simulation Programming System - An Introduction

DESCRIPTIVE NOTES (Type of report and inclusive dates)
Technical Report, 1971

AUTHOR(S) (First name, middle initial, last name)
George E. Heidorn

REPORT DATE December 1971	7a. TOTAL NO. OF PAGES 56	7b. NO. OF REFS 10
8a. CONTRACT OR GRANT NO. b. PROJECT NO. PO 1-0177 c. Identifying Number NR 049-314 d.	9a. ORIGINATOR'S REPORT NUMBER(S) NPS-55HD71121A 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	

10. DISTRIBUTION STATEMENT
Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Information Systems Program, Office of Naval Research

13. ABSTRACT

A simulation programming system with which models for simple queuing problems can be built through natural-language interaction with a computer is described. In this system the English statement of a problem is first translated into a language-independent entity-attribute-value information structure, which can then be translated back into an equivalent English description and into a GPSS simulation program for the problem. This processing is done on an IBM 360/67 by a FORTRAN program which is guided by a set of stratified decoding and encoding rules written in a grammar-rule language developed for this system. A detailed example of the use of the system is included.

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
digital computer simulation						
simulation programming						
GPSS						
queuing problems						
computational linguistics						
natural language						
stratificational grammar						
grammar rule language						
mechanical translation						
semantics						
artificial intelligence						